

**EGE UNIVERSITY**

**FACULTY OF ENGINEERING**

**COMPUTER ENGINEERING DEPARTMENT**

**204 DATA STRUCTURES (3+1)**

**2021–2022 FALL SEMESTER**

**PROJECT-4 REPORT**

**(GRAPHS, GRAPH ALGORITHMS, TREES and OTHER SUBJECTS)**

**DELIVERY DATE**

22/01/2022

**PREPARED BY**

05190000114 Mahmut ÇELİK

05190000027 Özgür BAYRAŞA

İçindekiler

[1.a AVL ağacına istenilen değerlerin eklenerek oluşan yeni AVL ağaçlarının çizimleri 3](#_Toc93185210)

[1.b Heapa istenilen değerin eklendikten ve silme işlemi yapıldıktan sonra heapin çizimi 4](#_Toc93185211)

[2.a B-Tree ekleme method (ya da AVL-Tree ekleme method/Red-Black tree/Huffman encoding tree) kodu 4](#_Toc93185212)

[2.b B-Tree ekleme kodunun adım adım açıklanması 12](#_Toc93185213)

[3.a Dijkstra kodu ve yapılan testler 15](#_Toc93185214)

[3.b Prim MST kodu ve yapılan testler 24](#_Toc93185215)

[3.c BFT ya da DFT kodu ve yapılan testler 33](#_Toc93185216)

[3.d Verilen Big-O tablosunun doldurulmuş hali 41](#_Toc93185217)

[4.a Verilen çizgenin python ortamında grafiksel olarak oluşturan kod 41](#_Toc93185218)

[4.b Verilen çizgede en kısa yol bulma sonuçları 44](#_Toc93185219)

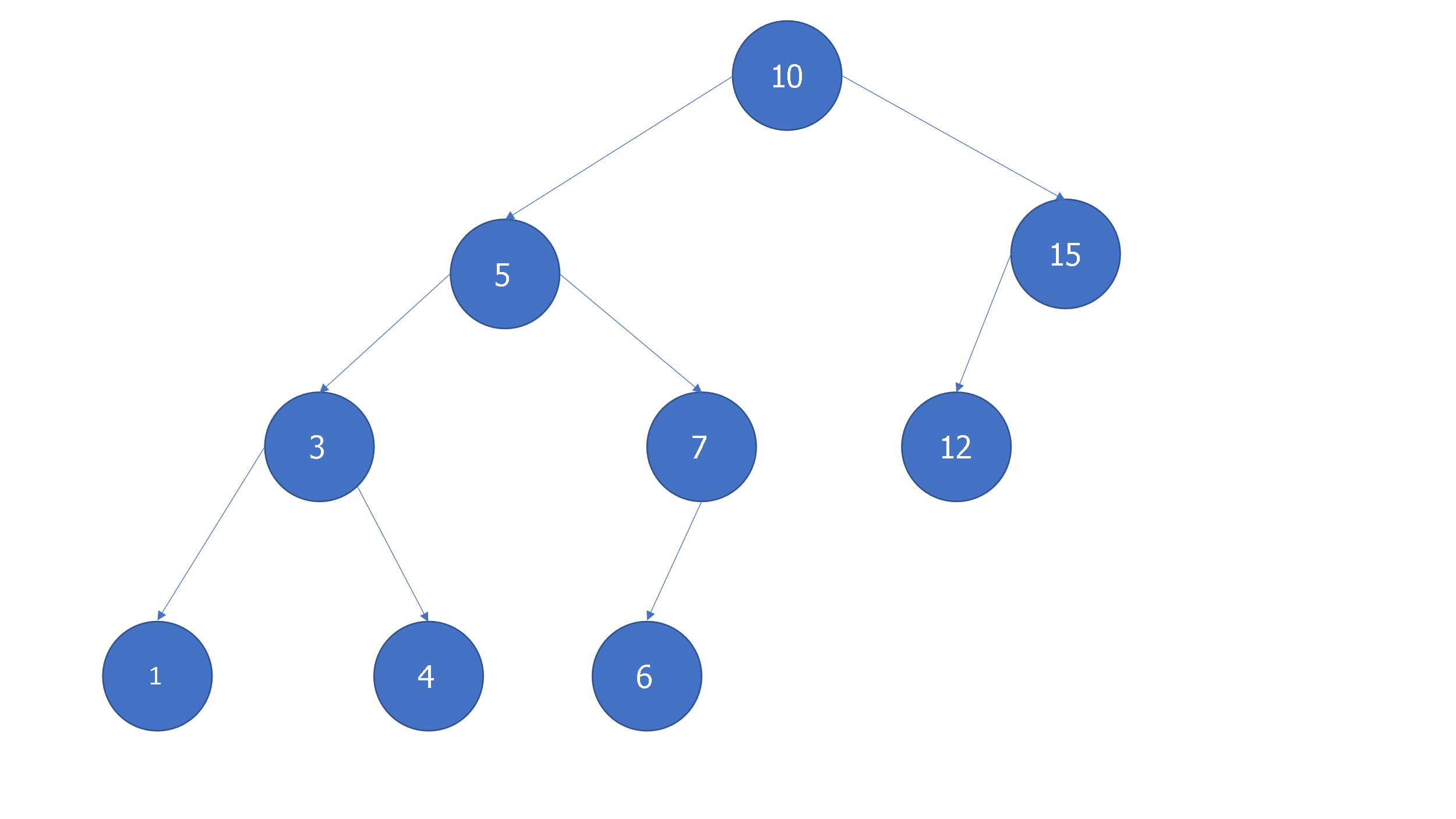
[5.a. Prim ve Kruskal algoritması karşılaştırması 44](#_Toc93185220)

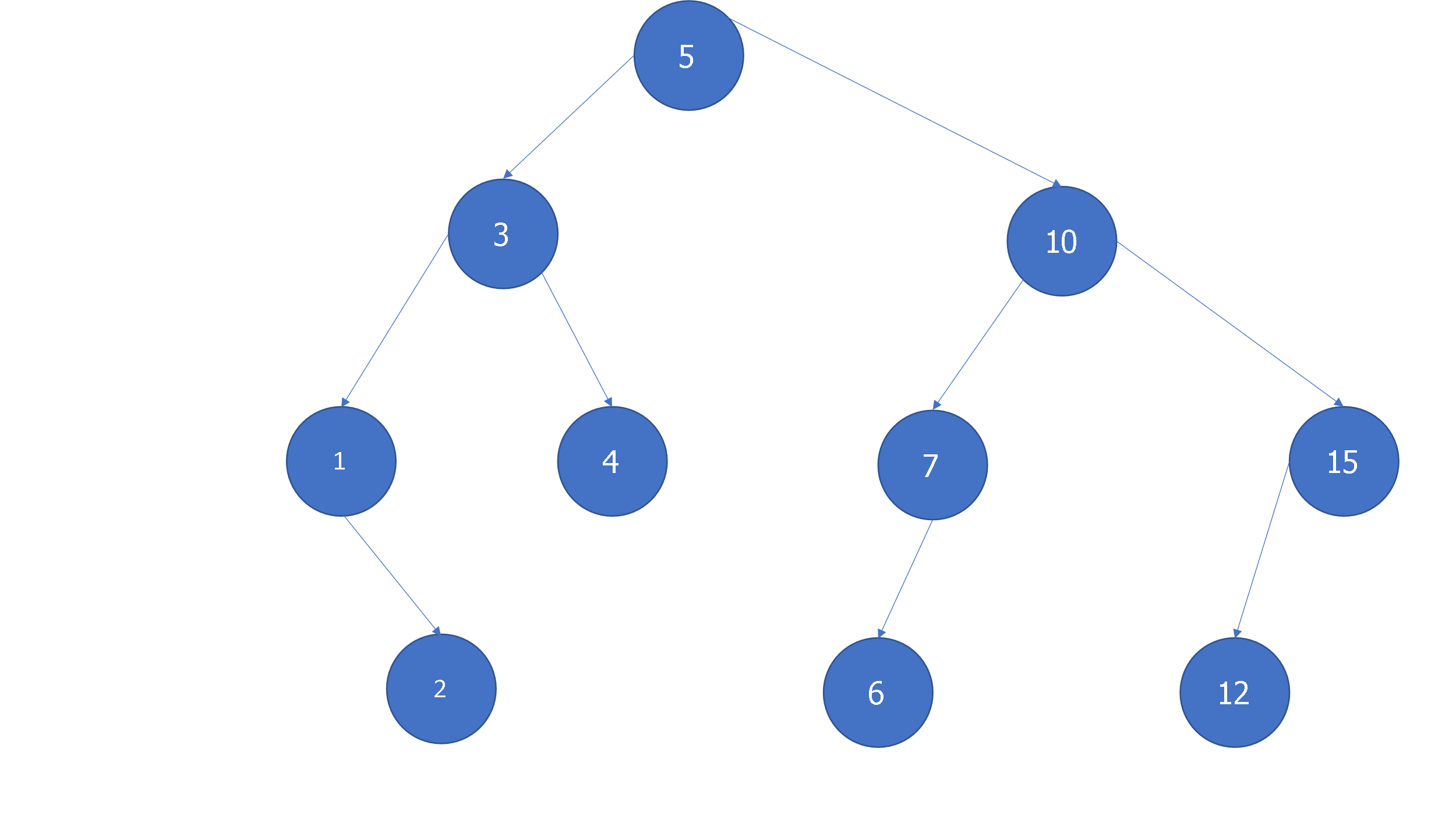
[5.b Verilen çizgede en kısa yol bulma sonuçları 46](#_Toc93185221)

[Özdeğerlendirme Tablosu 49](#_Toc93185222)

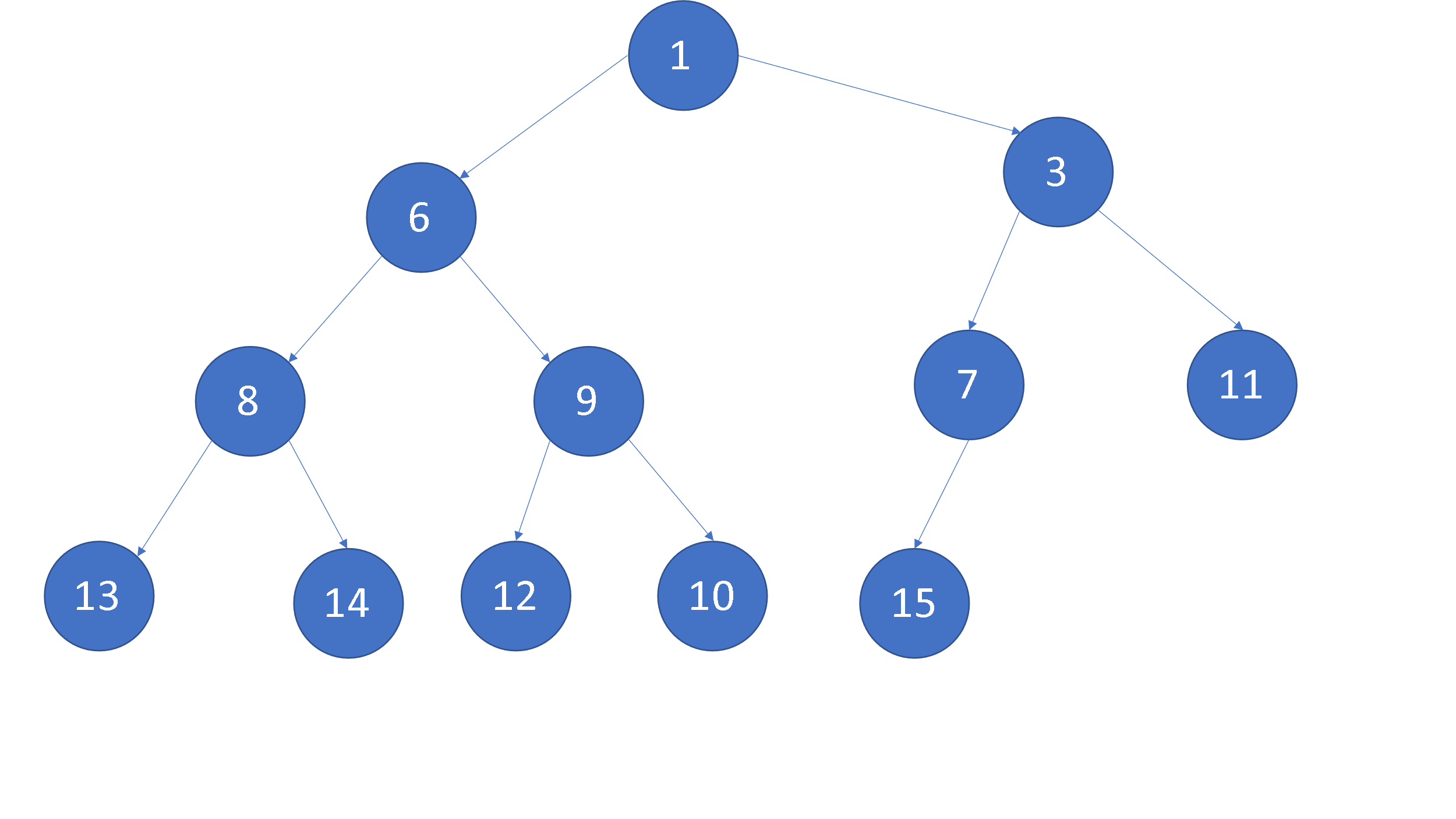
GRAPHS, GRAPH ALGORITHMS, TREES and OTHER SUBJECTS

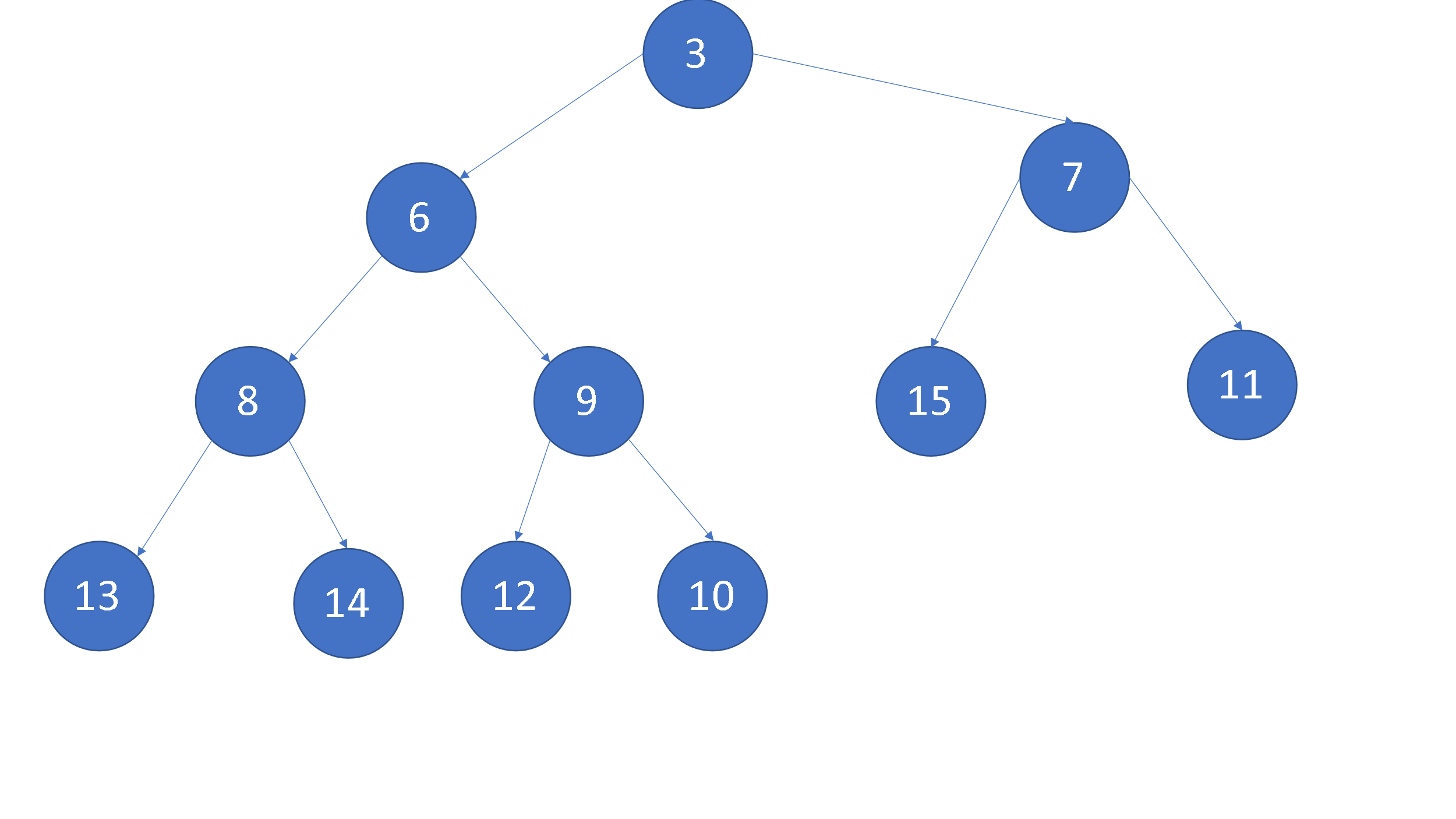
## 1.a AVL ağacına istenilen değerlerin eklenerek oluşan yeni AVL ağaçlarının çizimleri





## 1.b Heapa istenilen değerin eklendikten ve silme işlemi yapıldıktan sonra heapin çizimi





## 2.a B-Tree ekleme method (ya da AVL-Tree ekleme method/Red-Black tree/Huffman encoding tree) kodu

AVL ICIN GEREKLI CLASS

// AVL AĞACINA Ekleme Yapılan Java Programı

class Node {

int key, height;

Node left, right;

Node(int d) {

key = d;

height = 1;

}

// Daha güzel bir bastırılma için toString metodu tanımlanmıştır.

@Override

public String toString() {

return "Key: " +key ;

}

}

class AVLTree {

Node root;

// Ağacın boyunu döndüren metot tanımlanmıştır.

int height(Node N) {

if (N == null)

return 0;

return N.height;

}

// İki adet tamsayıdan maksimum olanı döndüren metot tanımlanmıştır.

int max(int a, int b) {

return (a > b) ? a : b;

}

// y köküne sahip olan ağacın sağ kısmını dengeleyen metottur.

Node rightRotate(Node y) {

Node x = y.left;

Node T2 = x.right;

// Dönüştürmenin yapılması (dengelenme).

x.right = y;

y.left = T2;

// Yüksekliklerin güncellenmesi.

y.height = max(height(y.left), height(y.right)) + 1;

x.height = max(height(x.left), height(x.right)) + 1;

// Yeni kökü döndür.

return x;

}

// y köküne sahip olan ağacın sağ kısmını dengeleyen metottur.

Node leftRotate(Node y) {

Node x = y.right;

Node T2 = x.left;

// Dönüştürmenin yapılması (dengelenme).

x.left = y;

y.right = T2;

// Yüksekliklerin güncellenmesi.

y.height = max(height(y.left), height(y.right)) + 1;

x.height = max(height(x.left), height(x.right)) + 1;

// Yeni kökü döndür.

return x;

}

// N köküne sahip ağacın dengesini bulan metottur.

// (SOL YÜKSEKLİK - SAĞ YÜKSEKLİK)

int getBalance(Node N) {

if (N == null)

return 0;

return height(N.left) - height(N.right);

}

// AVL Ağacına ekleme yapılabilen metot tanımlanmıştır.

// Kendisine verilen ağaca ekleme yapılır ve yeni ağaç döndürülür.

Node insert(Node rootNode, int key) {

// 1. Kök boş ise, yeni kök oluştur ve döndür (Köke ekle).

if (rootNode == null)

return (new Node(key));

// Kök boş değilse uygun çocuğa ekleme yap.

if (key < rootNode.key)

rootNode.left = insert(rootNode.left, key);

else if (key > rootNode.key)

rootNode.right = insert(rootNode.right, key);

else // Eşit anahtar için ekleme yapılmaz, aynı ağaç döndürülür.

return rootNode;

/\* 2. Eklenme olduğu için yüksekliği güncelle. \*/

rootNode.height = 1 + max(height(rootNode.left),

height(rootNode.right));

/\* 3. Dengesizlik durumunu kontrol et. Sol, sağ yükseklik arasında

bir birimden fazla fark varsa dengeleme yap. (AVL TREE)\*/

int balance = getBalance(rootNode);

// Eklenme durumunda dengesizlik olduysa dört durum olabilir.

// 1- Solun soluna ekleme yapılması.

if (balance > 1 && key < rootNode.left.key)

{System.out.println(rootNode);

return rightRotate(rootNode);

}

// 2- Sağın sağına ekleme yapılması.

if (balance < -1 && key > rootNode.right.key)

return leftRotate(rootNode);

// 3- Solun sağına ekleme yapılması.

if (balance > 1 && key > rootNode.left.key) {

rootNode.left = leftRotate(rootNode.left);

return rightRotate(rootNode);

}

// 4- Sağın soluna ekleme yapılması.

if (balance < -1 && key < rootNode.right.key) {

rootNode.right = rightRotate(rootNode.right);

return leftRotate(rootNode);

}

// Düğümü döndür.

return rootNode;

}

// Ağacın preorder dolaşılarak bastırılması.

void preOrder(Node node) {

if (node != null) {

System.out.print(node.key + " ");

preOrder(node.left);

preOrder(node.right);

}

}

}

MAIN CLASS

public class Proje4 {

public static void main(String[] args) {

AVLTree tree = new AVLTree();

/\* AVL AĞACINA Ekelemeler yapılarak test yapılmaktadır.

İlk soruda önce 3 sonra da 2 eklenen ağacın da eklenmesi test

edilmektedir.

\*/

tree.root = tree.insert(tree.root, 10);

tree.root = tree.insert(tree.root, 5);

tree.root = tree.insert(tree.root, 15);

tree.root = tree.insert(tree.root, 4);

tree.root = tree.insert(tree.root, 7);

tree.root = tree.insert(tree.root, 12);

tree.root = tree.insert(tree.root, 1);

tree.root = tree.insert(tree.root, 6);

tree.root = tree.insert(tree.root, 3); // Denge bozuluyor.

/\* Eklenme sonrasında ilk soruda çizilen AVL Ağacına göre çıkması

gereken sonuç, ağaç preOrder dolaşıldığında;

10 5 3 1 4 7 6 15 12 olacaktır.

\*/

System.out.println("Ağacın preorder dolaşılması" +

" sonucunda ortaya çıkan AVL Ağacı: ");

tree.preOrder(tree.root);

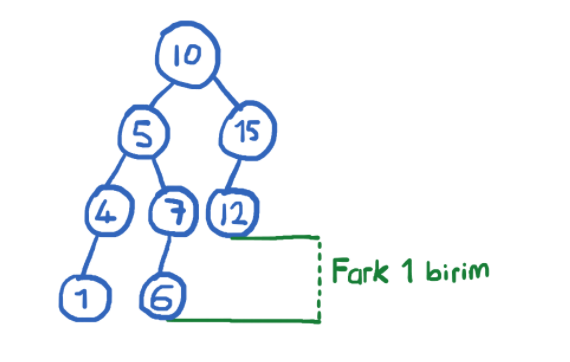
}

}

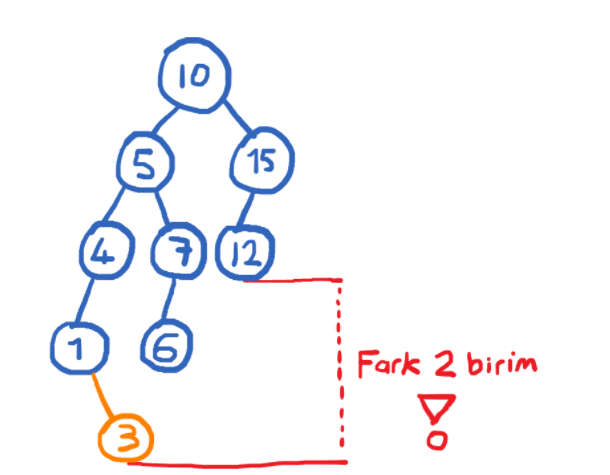
## 2.b B-Tree ekleme kodunun adım adım açıklanması

Yapılan işlem şu şekildedir;

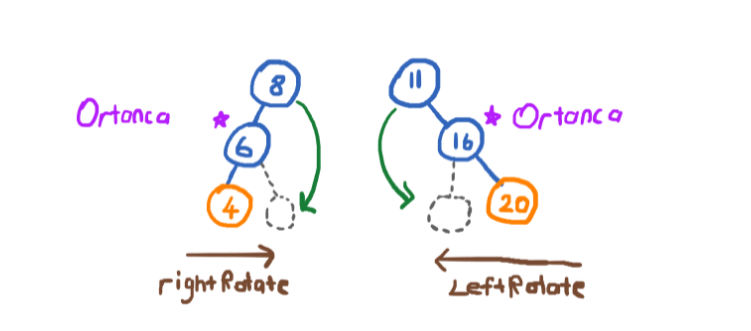
1. Öncelikle 10, 5, 15, 4, 7, 12, 1, 6 değerleri sırasıyla AVL ağacına normal bir şekilde eklenir. Düğümlerin sol çocukları, kendilerinden küçük tamsayılar, sağ çocukları ise kendilerinden büyük tamsayılardır.



1. AVL ağacında her düğümdeki sol ve sağ ağaç arasındaki yükseklik farkı en fazla bir olabilir. Az önceki eklemelerde de görüldüğü gibi bu durum bozulmamıştır. **Ancak 3 değerinin eklenmesi AVL Ağacının koşulunu bozacaktır.**



1. AVL ağacının insert metodunda denge bozulduğu için dengelenme işlemi yapılmalıdır. 1 değerine eklenme yapıldığına dengesizlik olmuştur. Dolayısıyla 1 değeri, torunu olan 4 değeri ve kendisine 1 değerine eklenen 3 değeri arasında **dengeleme işlemi yapılabilir.**
   1. Dengeleme işleminde duruma göre hamleler yapılır. Örneğin 8 - 6 - 4 düğümü ve 11 - 16 - 20 düğümleri gibi, **sadece tek tarafa ekleme yapılan** düğümler için tek hamle yeterlidir. Yoğunluk solda ise sağa dengeleme (rightRotate), yoğunluk sağda ise sola dengeleme (leftRotate) işlemi yapılmalıdır. İşlemlerin mantığında en tepedeki düğüm, kendi çocuğunun boş olan bölgesine yerleşir. En tepedeki düğümün üst düğümü varsa da bu üst düğüm artık tepedeki düğümün çocuğunu gösterecektir.

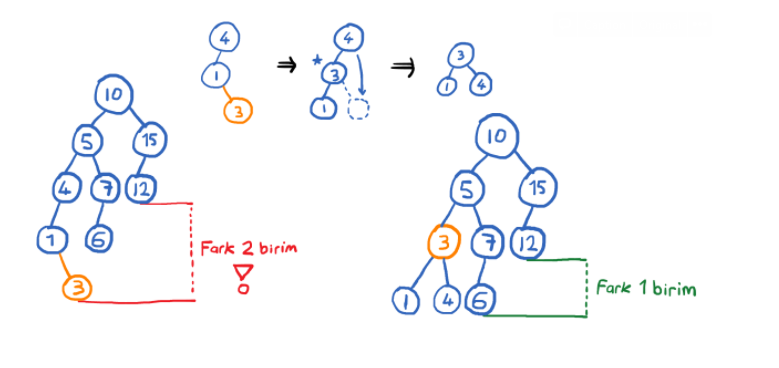


* 1. Fakat her zaman **tek yönde ekleme olarak bozulma yaşanmaz.** 8 - 6 - **10** ve 11 - 16 - **13** düğümlerini düşünelim. Burada ilkinde önce sol sonra sağa ekleme, ikincide ise önce sağ sonra sola eklenme vardır. Burada ise yapılması gereken basittir. **Yönü tek taraflı yapmak.** Örneğin 8 - 6 - 10 düğümlerinde önce sola sonra sağa eklenme var. Bunu tek sola yapmamız gerekecek, çünkü 8’ün soluna eklenme yapılmış. 6 değerinin sağındaki 10 değerini sola alıp iki değerin yerini değiştirmek olayı çözecektir (**yani leftRotate işlemi)** . **İlk adımdaki değişimden sonra düğüm yapısı 8 - 10 - 6 şeklinde ve sadece sola eklenmiş şekildedir.** 8’in solunda 10 değeri olması yanlıştır ancak ikinci adımdan sonra da bu durum düzelecektir. İkinci adımda ise normal tek yöndeki durum için yaptığımız işlemi yaparız. Ortanca değer (10 düğümü) sabit. Onun tepesindeki düğüm (8 düğümü) 10 düğümünün sağına yerleşecek. Dengelenme işlemi böylece hatasız şekilde tamamlanır. **11 - 16 - 13 düğüm yapısı için ise 8 - 6 - 10’daki işlemlerin ters yönlüsü yapılır.**

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

1. Şimdi 4 - 1 - 3 düğümlerinin dengelenmesini görsel olarak görelim. Anlatımdaki 3b maddesindeki gibi önce tek yönde yoğunluk sağlanır. Sonra da tek yönde uygulanan dengeleme işlemi yapılır.



## 3.a Dijkstra kodu ve yapılan testler

public class MAINCLASS {

public static void main(String[] args) {

Graph theGraph = new Graph(); //GRAPH CLASSINDAN NESNE OLUSTURULMASI

theGraph.addVertex('Z'); //0 //GRAPH A VERTEX YANI KOSE EKLENMESI

theGraph.addVertex('Y'); //1

theGraph.addVertex('T'); //2

theGraph.addVertex('X'); //3

theGraph.addVertex('V'); //4

theGraph.addVertex('W'); //5

theGraph.addVertex('U'); //6

theGraph.addVertex('S'); //7

//GRAPH GECISLERININ EKLENMESI

theGraph.addEdge(0, 1, 5); //ZY 5

theGraph.addEdge(0, 2, 10); //ZT 10

theGraph.addEdge(1, 2, 5); //YT 5

theGraph.addEdge(1, 3, 3); //YX 3

theGraph.addEdge(1, 4, 4); //YV 4

theGraph.addEdge(2, 4, 8); //TV 8

theGraph.addEdge(2, 6, 3); //TU 3

theGraph.addEdge(2, 7, 4); //TS 4

theGraph.addEdge(3, 4, 1); //XV 1

theGraph.addEdge(3, 5, 2); //XW 2

theGraph.addEdge(4, 5, 1); //VW 1

theGraph.addEdge(4, 6, 3); //VU 3

theGraph.addEdge(5, 6, 1); //WU 1

theGraph.addEdge(6, 7, 2); //US 2

System.out.println("Shortest paths"); //DIJKTRA'S SHORTEST PATH

theGraph.path(); // shortest paths

System.out.println();

}

}

package graphs;

class Edge { //OKLARIN YANI GECISLER ICIN GEREKLI CLASS

public int srcVert; // EDGE IN BASLADIGI YANI HANGI DUGUMDEN CIKILDIGINI TUTAN DEGISKEN DUGUMUN INDEXINI TUTAR

public int destVert; //EDGE IN BITTIGI YANI HANGI DUGUME GIDILGINITUTAN DEGISKEN DUGUMUN INDEXINI TUTAR

public int distance; //IKI KOSE ARASI MESAFEYI TUTAR

public Edge(int srcVert , int destVert , int distance){ //CONSTRUCTOR ICINDE ESITLEMELER YAPILDI

this.srcVert = srcVert;

this.destVert = destVert;

this.distance = distance;

}

}

class Vertex { //Node ları olusturmak için gerekli class vertex de denebilir

public char label; //node ların adını tutmak için değişken

public boolean isInTree; //AGACTA BULUNUP NULUNMADIGINI BILMEK ICIN BOOLEAN DEGISKEN

public Vertex(char labelName){ //CONSTRUCTOR

label = labelName;

isInTree = false;

}

}

class Graph {

private final int MAX\_VERTS = 20; //OLUSABILECEK EN FAZLA NODE SAYISINI YANI VERTEX SAYISI 20 OLARAK ATANDI

private final int INFINITY = 1000000; //SONSUZLUK DEGERI OLUSTURULMASI

private Vertex vertexList[]; //NODELARI KOSELERI TUTACAK VE ICINDE VERTEX CLASSINDAN OLUSTURULMUS NESNELERI BARINDIRACAK DIZI

private int adjMat[][]; //KOMSULUK MATRISI

private int nVerts; // KAC TANE KOSE OLDUGUNU TUTACAK DEGISKEN

private int currentVert; //SUANKI KOSENIN INDEXI

private DistPar sPath[]; // array for shortest-path data

private int startToCurrent; // distance to currentVert

//MINIMUM SPANNING TREE ICIN GEREKLILER

private PriorityQueue thePQ;

private int nTree;

public Graph(){ //GRAPH CLASS I ICIN CONSTRUCTOR OLUSTURDUM

vertexList = new Vertex[MAX\_VERTS]; //KOSELERI TUTACAK DIZININ UZUNLUGU BELIRLENEREK BELLEKTE OLUSTURULDU

adjMat = new int[MAX\_VERTS][MAX\_VERTS]; //KOMSULUK MATRISI KOSELER ARASINDA KOMSULUK OLUP OLMADIGINI TUTMAK ICIN GEREKLI OLDUGUNDAN KOSE SAYISININ KARESI KADAR OLACAK

nVerts = 0; // TOPLAM KOSE SAYISINI ILK BASTA 0 ATADIM

for(int a = 0 ; a <MAX\_VERTS ; a++){ //IKI TANE FOR DONGUSUYLE KOMSULUK MATRISINDEKI ELEMANLARI DOLASIP SONSUZ ATADIM ATADIM

for(int b = 0 ; b <MAX\_VERTS ; b++){

adjMat[a][b] = INFINITY;

}

}

//MINIMUM SPANNING TREE ICIN GEREKLI

thePQ = new PriorityQueue();

//Dijkstra’s Shortest Path

sPath = new DistPar[MAX\_VERTS]; // shortest paths

}

public void addVertex(char charToAddGraph){

//VERTEX LISTE ELEMAN EKLENIR VE ELEMAN SAYISINI TUTAN DEGISKEN 1 ARTTIRILIR

vertexList[nVerts++] = new Vertex(charToAddGraph); //VERTEXTEN NESNE(KOSE) OLUSTURULUP EKLENIYOR

}

public void addEdge(int startVertex , int endVertex , int weight){ //EDGELERIN AGIRLIKLARI EKLENIYOR

adjMat[startVertex][endVertex] = weight;

adjMat[endVertex][startVertex] = weight;

}

public void displayVertex(int index){

System.out.print(vertexList[index].label);

}

//AGIRLIKLI GRAPH OLUSTURMA ICIN GEREKLI KISIM BITTI

//DIJKTRAS SHORTEST PATH

public void path() // SHORTEST PATH ALGORITMASININ GERCEKLESTIRILMESI ICIN GERKELI METOD

{

int startTree = 0; // 0.INDEXTEKI NODE DAN BASLAMASI ICIN DEGISKENE 0 ATADIM

vertexList[startTree].isInTree = true; //AGACIN BOS OLUP OLMADIGINI KONTROL ETME

nTree = 1;

// transfer row of distances from adjMat to sPath

for(int j=0; j<nVerts; j++) //ILK NODE ICIN UZAKLIK MATRISINDEKI UZAKLIKLAR SPATH ADINDAKI DIZIYE AKTARILIR

{

int tempDist = adjMat[startTree][j]; //MESAFE

sPath[j] = new DistPar(startTree, tempDist); //ESITLEME ISLEMI

}

// until all vertices are in the tree

while(nTree < nVerts) //BASTA OLUSTURULAN DEGISKENIN KOSE SAYISINDAN KUCUK OLDUGU DURUM KADAR CALISACAK WHILE DONGUSU

{

int indexMin = getMin();

int minDist = sPath[indexMin].distance; //YUKARIDA BELLI NODE ICIN EKLENMIS UZAKLIKLARDAN EN KUCUK OLANIN INDEXI ALINIR VE EN KUCUK MESAFEYI TUTACAK DEGISKENE ATANIR

if(minDist == INFINITY) //HEPSI SONSUZSA YANI NODE VAR FAKAT HICBIR NODELA BAGLANTISI YOKSA

{

System.out.println("BAGLANTI YOK");

break;

}

else

{

currentVert = indexMin; //YENI NODE A GECILDI

startToCurrent = sPath[indexMin].distance; //SUANKI NODE ICIN MIN MESAFE BULUNUR

}

// put current vertex in tree

vertexList[currentVert].isInTree = true; //BULUNAN YENI NODE GRAPHTA OLUP OLMADIGI KONTROL EDILIR VARSA DEGISKEN 1 ARTTIRILIR

nTree++;

adjust\_sPath(); //YOLDA DEGISIKLIKLER ICIN GUNCELLEME YAPILIR HER DONGGU SONUNDA GERCEKLESIR

}

displayPaths();

nTree = 0;

for(int j=0; j<nVerts; j++)

vertexList[j].isInTree = false; //DEGERLER FALSE A CEVRILIR

}

public int getMin() //EN KISA UZAKLIKTAKI KOSEYI BULMAMIZI SAGLAYAN METOD

{

int minDist = INFINITY;

int indexMin = 0;

for(int j=1; j<nVerts; j++) // TUM KOSELER DOLASILIR

{

if( !vertexList[j].isInTree &&

sPath[j].distance < minDist ) //GRAPHTA YOKSA VEYA DAHA KUCUKSE EN KISA MESAFE GUNCELLENIR

{

minDist = sPath[j].distance;

indexMin = j;

}

}

return indexMin; // MINIMUM UZAKLIGIN INDEXI GERI DONDURULUR

}

public void adjust\_sPath() //SUTUNLAR ICIN GUNCELLEME YAPAR

{

// adjust values in shortest-path array sPath

int column = 1; // skip starting vertex

while(column < nVerts) // go across columns

{

// if this column's vertex already in tree, skip it

if( vertexList[column].isInTree )

{

column++;

continue;

}

// calculate distance for one sPath entry

// get edge from currentVert to column

int currentToFringe = adjMat[currentVert][column];

// add distance from start

int startToFringe = startToCurrent + currentToFringe;

// get distance of current sPath entry

int sPathDist = sPath[column].distance;

// compare distance from start with sPath entry

if(startToFringe < sPathDist) // if shorter,

{ // update sPath

sPath[column].parentVert = currentVert;

sPath[column].distance = startToFringe;

}

column++;

} // end while(column < nVerts)

} // end adjust\_sPath()

public void displayPaths() //BULUNAN EN KISA YOLLARIN GOSTERIMI SECILEN 1 NODE DAN DIGER TUM NODELARA OLAN EN KISA YOLLAR GOSTERILIR

{

for(int j=0; j<nVerts; j++)

{

System.out.print(vertexList[j].label + "=");

if(sPath[j].distance == INFINITY) //BAGLANTI YOKSA YANI UZAKLIK SONSUZSA

System.out.print("inf");

else

System.out.print(sPath[j].distance); //MESAFENIN YAZDIRILMASI

char parent = vertexList[ sPath[j].parentVert ].label;

System.out.print("(" + parent + ") ");

}

System.out.println("");

}

} //GRAPH BITIMI



## 3.b Prim MST kodu ve yapılan testler

package graphs;

public class MAINCLASS {

public static void main(String[] args) {

Graph theGraph = new Graph(); //GRAPH CLASSINDAN NESNE OLUSTURULMASI

theGraph.addVertex('Z'); //0 //GRAPH A VERTEX YANI KOSE EKLENMESI

theGraph.addVertex('Y'); //1

theGraph.addVertex('T'); //2

theGraph.addVertex('X'); //3

theGraph.addVertex('V'); //4

theGraph.addVertex('W'); //5

theGraph.addVertex('U'); //6

theGraph.addVertex('S'); //7

//GRAPH GECISLERININ EKLENMESI

theGraph.addEdge(0, 1, 5); //ZY 5

theGraph.addEdge(0, 2, 10); //ZT 10

theGraph.addEdge(1, 2, 5); //YT 5

theGraph.addEdge(1, 3, 3); //YX 3

theGraph.addEdge(1, 4, 4); //YV 4

theGraph.addEdge(2, 4, 8); //TV 8

theGraph.addEdge(2, 6, 3); //TU 3

theGraph.addEdge(2, 7, 4); //TS 4

theGraph.addEdge(3, 4, 1); //XV 1

theGraph.addEdge(3, 5, 2); //XW 2

theGraph.addEdge(4, 5, 1); //VW 1

theGraph.addEdge(4, 6, 3); //VU 3

theGraph.addEdge(5, 6, 1); //WU 1

theGraph.addEdge(6, 7, 2); //US 2

System.out.print("Minimum spanning tree: ");

theGraph.MinimumSpanningTree();// PRIM'S MINIMUM SPANNING TREE

System.out.println("\n");

}

}

class Edge { //OKLARIN YANI GECISLER ICIN GEREKLI CLASS

public int srcVert; // EDGE IN BASLADIGI YANI HANGI DUGUMDEN CIKILDIGINI TUTAN DEGISKEN DUGUMUN INDEXINI TUTAR

public int destVert; //EDGE IN BITTIGI YANI HANGI DUGUME GIDILGINITUTAN DEGISKEN DUGUMUN INDEXINI TUTAR

public int distance; //IKI KOSE ARASI MESAFEYI TUTAR

public Edge(int srcVert , int destVert , int distance){ //CONSTRUCTOR ICINDE ESITLEMELER YAPILDI

this.srcVert = srcVert;

this.destVert = destVert;

this.distance = distance;

}

}

class Vertex { //Node ları olusturmak için gerekli class vertex de denebilir

public char label; //node ların adını tutmak için değişken

public boolean isInTree; //AGACTA BULUNUP NULUNMADIGINI BILMEK ICIN BOOLEAN DEGISKEN

public Vertex(char labelName){ //CONSTRUCTOR

label = labelName;

isInTree = false;

}

}

class Graph {

private final int MAX\_VERTS = 20; //OLUSABILECEK EN FAZLA NODE SAYISINI YANI VERTEX SAYISI 20 OLARAK ATANDI

private final int INFINITY = 1000000; //SONSUZLUK DEGERI OLUSTURULMASI

private Vertex vertexList[]; //NODELARI KOSELERI TUTACAK VE ICINDE VERTEX CLASSINDAN OLUSTURULMUS NESNELERI BARINDIRACAK DIZI

private int adjMat[][]; //KOMSULUK MATRISI

private int nVerts; // KAC TANE KOSE OLDUGUNU TUTACAK DEGISKEN

private int currentVert; //SUANKI KOSENIN INDEXI

private DistPar sPath[]; // array for shortest-path data

private int startToCurrent; // distance to currentVert

//MINIMUM SPANNING TREE ICIN GEREKLILER

private PriorityQueue thePQ;

private int nTree;

public Graph(){ //GRAPH CLASS I ICIN CONSTRUCTOR OLUSTURDUM

vertexList = new Vertex[MAX\_VERTS]; //KOSELERI TUTACAK DIZININ UZUNLUGU BELIRLENEREK BELLEKTE OLUSTURULDU

adjMat = new int[MAX\_VERTS][MAX\_VERTS]; //KOMSULUK MATRISI KOSELER ARASINDA KOMSULUK OLUP OLMADIGINI TUTMAK ICIN GEREKLI OLDUGUNDAN KOSE SAYISININ KARESI KADAR OLACAK

nVerts = 0; // TOPLAM KOSE SAYISINI ILK BASTA 0 ATADIM

for(int a = 0 ; a <MAX\_VERTS ; a++){ //IKI TANE FOR DONGUSUYLE KOMSULUK MATRISINDEKI ELEMANLARI DOLASIP SONSUZ ATADIM ATADIM

for(int b = 0 ; b <MAX\_VERTS ; b++){

adjMat[a][b] = INFINITY;

}

}

//MINIMUM SPANNING TREE ICIN GEREKLI

thePQ = new PriorityQueue();

//Dijkstra’s Shortest Path

sPath = new DistPar[MAX\_VERTS]; // shortest paths

}

public void addVertex(char charToAddGraph){

//VERTEX LISTE ELEMAN EKLENIR VE ELEMAN SAYISINI TUTAN DEGISKEN 1 ARTTIRILIR

vertexList[nVerts++] = new Vertex(charToAddGraph); //VERTEXTEN NESNE(KOSE) OLUSTURULUP EKLENIYOR

}

public void addEdge(int startVertex , int endVertex , int weight){ //EDGELERIN AGIRLIKLARI EKLENIYOR

adjMat[startVertex][endVertex] = weight;

adjMat[endVertex][startVertex] = weight;

}

public void displayVertex(int index){

System.out.print(vertexList[index].label);

}

//AGIRLIKLI GRAPH OLUSTURMA ICIN GEREKLI KISIM BITTI

//BUNDAN SONRASI DOLASMA ICIN GERKELI METODLAR

public void MinimumSpanningTree(){ //PRIMS MINIMUM SPANNING TREE ICIN GEREKLI METOD

currentVert = 0; //0.VERTEXTEN BASLANIR ALGORITMANIN MANTIGINDA HERHANGI BIR DUGUM SECILIR VE BU DUGUME EN YAKIN KOMSU SECILIR VE ADA OLUSUR. SONRASINDA OLUSAN ADAYA SIMDILIK 2VERTEXLIK KISIM ICIN EN YAKIN VERTEX

//BULUNUR EGER BULUNAN VERTEX CYCLE OLUSTURMUYORSA GECIS YAPILIR VE ADAYA EKLENMIS OLUR BU SEKILDE ILERLEYEREK TUM GRAPH ADA SEKLINE GETIRILIR VE GRAPHTAKI TUM VERTEXLERE ULASABILECEGIMIZ UYGUN EN KISA YOLLARDAN OLUSAN AGAC ELDE ETMIS OLURUZ

while(nTree < nVerts -1 ){ //TREEDEKI ELEMAN SAYISI VERTEX SAYISINDAN KUCUK OLDUKCA CALISACAK WHILE DONGUSU

vertexList[currentVert].isInTree = true; //AGACTA OLUP OLMADIGI KONTROL EDILIR

nTree++;

for(int a = 0 ; a<nVerts ; a++){ //VERTEXLER DOLASILIR VE UYGUN VERTEX IF BLOKLARINDA CONTINUE KULLANILARAK BULUNUR VE PRIORITYQUEUE A EKLENIR

if(a==currentVert) //SUANKI VERTEXE ESITSE GECILIR

continue;

if(vertexList[a].isInTree) //AGACTA VARSA GECILIR

continue;

int distance = adjMat[currentVert][a]; //SUANKI VERTEXLE ARASINDAKI MESAFE ALINIR

if(distance ==INFINITY) //SONSUZ MESAFEDEYSE ARALARINDA BAGLANTI YOKTUR GECILIR

continue;

putInPQ(a,distance); //TUM KOSULLARI SAGLADIGI ICIN PQ A EKLENIR

}

if(thePQ.size()==0){ //PQ NUN UZUNLUGU 0 SA HICBIR ELEMAN EKLENEMEMISTIR TUM VERTEXLER AYRI SEKILDEDIR

System.out.println("GRAPH BAGLI DEGIL");

return;

}

Edge theEdge = thePQ.removeMin(); //OLUSTURULAN PRIORITYQUEUE DEN EN KUCUK ELEMAN CEKILIR VE EDGE CLASSINDAN OLUSTURULMUS NESNEYE ATANIR

int sourceVert = theEdge.srcVert; //OKUN ARKA KISMI

currentVert = theEdge.destVert; //OKUN UC KISMI

System.out.print(vertexList[sourceVert].label); //YAZDIRILMA ISLEMI

System.out.print(vertexList[currentVert].label);

System.out.print(" ");

} //WHILE BITIS

for(int a = 0 ; a<nVerts ; a++){ //ISARETLER KALDIRILIYOR

vertexList[a].isInTree = false;

}

} //MINIMUM SPANNING TREE BITIS

} //GRAPH BITIMI

class DistPar // distance and parent

{ // items stored in sPath array

public int distance; // distance from start to this vertex

public int parentVert; // current parent of this vertex

// -------------------------------------------------------------

public DistPar(int pv, int d) // constructor

{

distance = d;

parentVert = pv;

}

// -------------------------------------------------------------

} // end class DistPar

class PriorityQueue{ //ONCELIK KUYRUGUNUN AVANTAJINI PRIMS MINIMUM SPANNING TREEDE KULLANABILMEMIZ ICIN GEREKLI PQ

private final int SIZE = 20; //GEREKLI DEGISKENLER

private Edge[] queArray;

private int size;

public PriorityQueue(){ //CONSTRUCTOR

queArray = new Edge[SIZE];

size = 0;

}

public void insert(Edge item){ //ELEMAN EKLEME METODU AZALAN PQ KULLANILACAGI ICIN GEREKLI FOR DONGULERINI BARINDIRIYOR

int j;

for(j=0 ; j<size ; j++){ //DIZI DOLASILIYOR VE YENI EKLENECEK ELEMANIN MESAFESI BUYUKSE FOR DONGUSUNDEN BREAK ILE CIKILIYOR

if(item.distance >= queArray[j].distance) break;

}

for(int k = size-1 ; k>=j; k--){ //DIZI OTELENIYOR 1SAGA

queArray[k+1] = queArray[k];

}

queArray[j] = item; //J.INDEXE YENI GELECEK ELEMAN EKLENIYOR

size++;

}

public Edge removeMin(){ //EN SONDAKI ELEMAN EN KUCUK OLDUGUNDAN EN SONDAKI SILINIR VE SIZE KOSELI PARANTEZ ICINDE 1 AZALTILIR SONRA SILINIR

return queArray[--size];

}

public void removeN(int n){ //GELEN INDEXTEKI ELEMANI SILEN VE SONRASINDA DIZIDEKI ELEMANLARI OTELEYEN METOD

for(int j=n ; j<size-1 ;j++){

queArray[j] = queArray[j+1];

}

size--;

}

public Edge peekMin(){ //EN KUCUK ELEMANI GONDERECEK METOD AZALAN ONCELIKTE YANI BUYUKTEN KUCUGE BIR PQ KULLANILACAGI ICIN SONUNCU ELEMAN GONDERILIR

return queArray[size-1];

}

public int size(){ //SIZE I GONDERECEK METOD

return size;

}

public boolean isEmpty(){ //ISEMPTY METODU

return (size==0);

}

public Edge peekN(int n){ //GELEN INDEX DEGERINDEKI ELEMANIN GERI GONDERILMESI SILME YAPILMAZ

return queArray[n];

}

public int find(int findDex){ //GELEN ELEMANIN INDEXINI GONDERECEK FIND METODU

for(int a = 0 ; a<size ; a++){

if(queArray[a].destVert == findDex) return a;

}

return -1;

}

}



## 3.c BFT ya da DFT kodu ve yapılan testler

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package proje.pkg4;

/\*\*

\*

\* @author Özgür Bayraşa

\*/

// StackX Sınıfı tanımlanmıştır.

public class StackX

{

// Stack'in boyutu 20 birimdir. Bunun dışında özellikler de tanımlanmıştır.

private final int SIZE = 20;

private int[] st; // Stack dizisi.

private int top; // Tepedeki eleman.

// ------------------------------------------------------------

// Yapılandırıcı metot tanımlanmıştır.

public StackX()

{

st = new int[SIZE]; // Dizi boyut kadar oluşturulmuştur.

top = -1; // Henüz eleman olmadığı için tepedeki değer -1'dir.

}

// ------------------------------------------------------------

// Stack'e eleman eklenen metot tanımlanmıştır.

public void push(int j)

{

st[++top] = j; // Tepe değer bir arttırılır ve eleman eklenir.

}

// ------------------------------------------------------------

// Stack'ten eleman alınan metot tanımlanmıştır.

public int pop()

{

return st[top--]; // Tepedeki eleman döndürülür ve tepe indeks

// azalrılır.

}

// ------------------------------------------------------------

// Tepedeki elemanı döndüren metottur.

public int peek()

{

return st[top];

}

// ------------------------------------------------------------

// Stack'te elemena olup olmadığını kontrol eden metottur.

public boolean isEmpty()

{

return (top == -1);

}

// ------------------------------------------------------------

} // StackX sınıfınını sonu.

////////////////////////////////////////////////////////////////

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package proje.pkg4;

/\*\*

\*

\* @author Özgür Bayraşa

\*/

// Vertex sınıfı tanımlanmıştır.

public class Vertex

{

public char label; // Etiket

public boolean wasVisited; // Ziyaret edildi mi?

// ------------------------------------------------------------

public Vertex(char lab) // Yapılandırıcı metot.

{

label = lab;

wasVisited = false;

}

// ------------------------------------------------------------

} // Vertex sınıfının sonu.

////////////////////////////////////////////////////////////////

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package proje.pkg4;

/\*\*

\*

\* @author Özgür Bayraşa

\*/

// Graph sınıfı tanımlanmıştır.

class Graph

{

private final int MAX\_VERTS = 20; // Maksimum 20 verteks.

private Vertex vertexList[]; // Vertekslerin dizisi.

private int adjMat[][]; // Komşuluk matriksi

private int nVerts; // Anlık verteks sayısı.

private StackX theStack; // Stack

// ------------------------------------------------------------

public Graph() // Yapılandırıcı metot.

{

vertexList = new Vertex[MAX\_VERTS]; // Vertex listesi oluşturlur.

adjMat = new int[MAX\_VERTS][MAX\_VERTS]; // Komşuluk matrisi oluşturulur.

nVerts = 0;

for(int y=0; y<MAX\_VERTS; y++)

for(int x=0; x<MAX\_VERTS; x++)

adjMat[x][y] = 0; // Komşuluk Matrisi 0'dır.

theStack = new StackX();

}

// ------------------------------------------------------------

// Verteks ekleme metodu tanımlanmıştır.

public void addVertex(char lab)

{

// Verkex listesine eleman eklenir. Eleman sayısı arttırılır.

vertexList[nVerts++] = new Vertex(lab);

}

// ------------------------------------------------------------

// Kenar ekleme metodu tanımlanmıştır.

public void addEdge(int start, int end)

{

// İki kenar arası 1 olarak değiştirilir.

adjMat[start][end] = 1;

adjMat[end][start] = 1;

}

// ------------------------------------------------------------

// Verteksin bastırıldığı metot tanımlanmıştır.

public void displayVertex(int v)

{

System.out.print(vertexList[v].label);

}

// ------------------------------------------------------------

// dfs ile dolaşma metodu tanımlanmıştır.

public void dfs()

{

// 0. Verteksten başlanır.

vertexList[0].wasVisited = true; // Verteks işaretlenir.

displayVertex(0); // Bastırılır.

theStack.push(0); // Verteks stack'e eklenir.

while( !theStack.isEmpty() ) // Stack dolu iken...

{

// Stack'in tepesindeki verteksin komşusu akınır.

int v = getAdjUnvisitedVertex( theStack.peek() );

if(v == -1) // İlgili verteks yoksa...

theStack.pop();

else // İlgili verteks varsa...

{

vertexList[v].wasVisited = true; // İşaretle.

displayVertex(v); // Bastır.

theStack.push(v); // Stack'e Ekle.

}

}

// Stack boş olunca resetleme yap.

for(int j=0; j<nVerts; j++)

vertexList[j].wasVisited = false;

}

// ------------------------------------------------------------

// Verteksin komşusunu döndüren metot.

public int getAdjUnvisitedVertex(int v)

{

// Verteksin komşusu bulunursa ve işaretlenmemişse verteksi döndür.

// Bulunmazsa veya ziyaret edilmişse -1 döndür.

for(int j=0; j<nVerts; j++)

if(adjMat[v][j]==1 && vertexList[j].wasVisited==false)

return j;

return -1;

}

// ------------------------------------------------------------

} // Graph sınıfının sonu.

////////////////////////////////////////////////////////////////

package proje.pkg4;

public class Proje4 {

public static void main(String[] args) {

Graph theGraph = new Graph(); // Graph oluştulur.

theGraph.addVertex('A'); // 0

theGraph.addVertex('B'); // 1

theGraph.addVertex('C'); // 2

theGraph.addVertex('D'); // 3

theGraph.addVertex('E'); // 4

theGraph.addEdge(0, 1); // AB

theGraph.addEdge(1, 2); // BC

theGraph.addEdge(0, 3); // AD

theGraph.addEdge(3, 4); // DE

System.out.print("\nVisits: ");

theGraph.dfs(); // DFS

System.out.println();

}

}



## 3.d Verilen Big-O tablosunun doldurulmuş hali

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Dijkstra’s SP** | Prim’s MST | BFT | **Heap Insertion** |
| **Big-O** (Zaman Karmaşıklığı  Big-O Notasyonuna Göre) | NORMAL DURUMDA O(V2)  MIN PRIORITY QUEUE KULLANILARAK  O(V + E. logV)  DUSER | NORMAL DURUMDA O(V2)  PRIORITYQUEUE KULLANILARAK O(E.logV)  FIBONACCI HEAP KULLANILARAK O(E+VlogV) YE DUSER | TUM ELEMANLAR BIR KEZ DOLASILACAGI ICIN  O(N) | O(N.logN) |

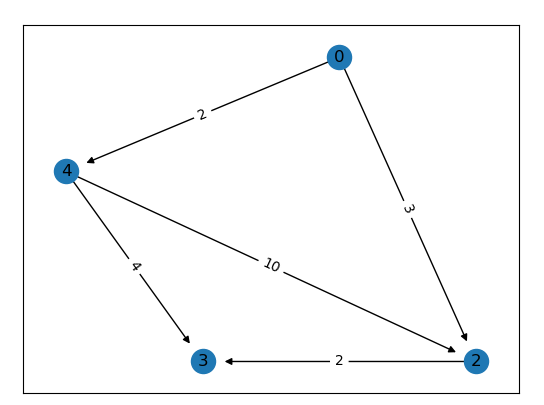
## 4.a Verilen çizgenin python ortamında grafiksel olarak oluşturan kod

import networkx as nx *#GEREKLI KUTUPHANELERIN IMPORT EDILMESI*import matplotlib.pyplot as plt  
  
def addNodes(G1): *#NODE LARI VE EDGELERI AYRI SEKILDE EKLEDIM CUNKU BU SEKILDE NODE LARIN KOORDINAT DEGERLERINI BELIRLEYEBILIYORUM EDGE EKLEYEREK OLUSAN NODELARDA KOORDINAT BELIRLEMEDE SIKINTI OLUSTU* G1.add\_node(0, pos=(3, 2)) *#NODELAR EKLENIR VE POS ICERISINDE KOORDINAT DEGERLERI ICERIR* G1.add\_node(1, pos=(6, 0.5))  
 G1.add\_node(2, pos=(4.5, -2))  
 G1.add\_node(3, pos=(1.5, -2))  
 G1.add\_node(4, pos=(0, 0.5))  
  
def addEdges(G1): *# NODELAR ARASINDAKI OKLARIN OLUSTURULMASI VE AGIRLIK ATTRIBUTELERININ EKLENMESI* G1.add\_edges\_from([(0, 4)], weight=2)  
 G1.add\_edges\_from([(0, 1)], weight=5)  
 G1.add\_edges\_from([(0, 2)], weight=3)  
 G1.add\_edges\_from([(1, 2)], weight=2)  
 G1.add\_edges\_from([(1, 3)], weight=6)  
 G1.add\_edges\_from([(2, 1)], weight=1)  
 G1.add\_edges\_from([(2, 3)], weight=2)  
 G1.add\_edges\_from([(4, 1)], weight=6)  
 G1.add\_edges\_from([(4, 2)], weight=10)  
 G1.add\_edges\_from([(4, 3)], weight=4)  
  
""" # YUKARIDA EDGE EKLERKEN CURWED ATTRIBUTE EKLEYIP BIRBIRLERI ARASINDA 1 DEN FAZLA OK VARSA CURWED TRUE YAPMAYI VE ONA GORE YAZDIRMAYI DUSUNDUM  
 #FAKAT ASAGIDAKI DONGU BEKLEDIGIM GIBI CALISMADI OKLAR UST USTE DENK GELIYOR   
 for edge in G1.edges.items():  
 for edge2 in G1.edges.items():  
 if(edge[0][0] == edge2[0][1] & edge[0][1] == edge2[0][0]):  
   
"""  
  
  
def makeChanges(G1): *#NODELAR GECISLER VE OKLARLA ILGILI OZELLIKLERIN BILGILERIN OLUSTURULDUGU KISIM* options = { *#BURADA OKLA ILGILI OZELLIKLER BELIRLENIR* 'node\_color': 'blue',  
 'node\_size': 1000,  
 'width': 1,  
 'arrowstyle': '-|>',  
 'arrowsize': 10,  
 }  
  
 node\_pos = nx.get\_node\_attributes(G1, 'pos') *#OLUSAN NODE UN KOORDINAT DEGERI ICIN GEREKLI DEGISKENIN ALINMASI* edge\_labels = dict([((u, v,), d['weight'])  
 for u, v, d in G1.edges(data=True)]) *# GECISLERDEKI BILGILERIN DICTIONARY E AKTARILMASI* show(G1 , node\_pos , edge\_labels , options) *#BASKIYA GECILMESI ICIN GEREKLI METOD*def show(G1 , node\_pos , edge\_labels , options): *#BASKININ YAPILACAGI METOD* nx.draw\_networkx\_nodes(G1, node\_pos) *#NODE LAR BASTILIR* nx.draw\_networkx\_labels(G1, node\_pos) *#ETIKETLERI EKLENIR 0 1 2 3 4* nx.draw\_networkx\_edges(G1, node\_pos, connectionstyle="arc3,rad = 0.0", arrows=True, \*\*options) *# OKLAR EKLENIR rad ı 0.3 yaparak denedim fakat hepsi bukuldugu ıcın attribute tanımlamam gerekli o kısmı yapamadım* nx.draw\_networkx\_edge\_labels(G1, node\_pos, edge\_labels=edge\_labels, arrows=True, \*\*options) *#OKLARIN AGIRLIK DEGERLERI EKLENIR* plt.show() *#KULLANICIYA GOSTERILMESI ICIN*def draw(G1): *#NODELARIN EDGELERIN DEGISIKLIKLERIN YAPILIP SONRASINDA BASKININ YAPILMAYA GECILMESINI SAGLAYAN METOD* addNodes(G1)  
 addEdges(G1)  
 makeChanges(G1) *#METODLARA GIDILIR VE ISLEMLER GERCEKLESTIRILIR* return G1  
  
def findShortestPath(G1): *# EN KISA YOLUN BULUNMASI ICIN METOD* number = 4 *#DOKUMANDA 4 SAYISI ICIN ISTENDIGI ICIN YAPTIM KODUN EN BASINDA DA YAPABILIRDIM AMA BURADA TANIMLADIM* for node in G1.nodes: *#NODELAR FOR DONGULERIYLE DOLASILIR* if(node != number): *#G1 ICINDEN GELEN NODE SAYIYA ESIT DEGILSE YANI KENDISI DEGILSE CALISACAK IF BLOGU* try: *#TRY CATCH KULLANDIM CUNKU 2 NODE ARASI YOL YOKSA NETWORKXNOPATH HATASI VERDIGI ICIN BUNU ONLEDIM* print("4.NODE ILE ",node,".NODE ARASINDAKI EN KISA YOL: ",nx.dijkstra\_path(G1, source=number, target=node, weight='weight')) *# EN KISA YOLUN DIJKSTRA\_PATH KULLANILARAK HESAPLANMASI* except nx.NetworkXNoPath :  
 print("4.NODE ILE ",node,".NODE ARASINDA GIDILEBILECEK YOL YOK") *#IKI NODE ARASI YOL OLMAYINCA CALISACAK METOD*def delete(G1): *#NODE SILMEK ICIN GEREKLI ISLEMLERI YAPAN METOD* number = 1 *#DOKUMANDA SILINMESI ICIN 1.NODE ISTENMIS BURADA TANIMLADIM DOKUMANIN ENBASINDA DA TANIMLAYABILIRDIM* G1.remove\_node(1) *# G1 DEN 1 NODE UNUN SILINMESI* makeChanges(G1) *#YUKARIDA YAPILAN CIZIM ICIN GEREKLI ISLEMLER TEKRAR YAPILIR VE METOD ICINDEKI SHOW ILE BASTIRILIR*def main(): *#MAIN METODU* G1 = nx.DiGraph() *#KUTUPHANEDEN DIGRAPH OLUSTURULDU* draw(G1) *#CIZDIRILMESI 4.SORU 1.BOLUM ICIN* findShortestPath(G1) *# EN KISA YOLUN BULUNMASI 4.SORU 2.BOLUM ICIN* delete(G1) *# NODE UN SILINMESI 4.SORU 3.BOLUM ICIN*main() *#MAIN METODUNUN CAGIRILMASI*

## 4.b Verilen çizgede en kısa yol bulma sonuçları

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu



## 5.a. Prim ve Kruskal algoritması karşılaştırması

Prim’in algoritmasında, ağaç tek bir rasgele düğüm ile başlar ve her döngüde bu düğümden itibaren genişler. Bu nedenle, algoritmanın düzgün çalışması için grafiğin bağlı bir grafik olması gerekir. Zaman karmaşıklığı O(V2) dir.

Kruskal’ın algoritmasında, algoritma en az ağırlıklı kenarla başlar ve her döngüde her bir kenarı seçmeye devam eder. Bu nedenle, algoritmada grafiğin bağlanması gerekmez. Zaman karmaşıklığı O (logV)

Prim'in algoritması bir düğümle başlarken Kruskal'ın algoritması bir kenarla başlar.

Prim algoritmasında, grafik bağlı bir grafik olmalı, Kruskal'lar ise bağlantısı kesilmiş grafiklerde de çalışabilir.

Prim'in algoritmaları bir düğümden diğerine yayılırken, Kruskal'ın algoritması kenarları kenarın konumu son adıma dayanmayacak şekilde seçer.

## 5.b Verilen çizgede en kısa yol bulma sonuçları

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

## Özdeğerlendirme Tablosu

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Points** | **Estimated Grade** | **Explanation** |
| **1 a) AVL Tree** | **10** | **10** |  |
| **1 b) Heap** | **10** | **10** |  |
| **2) B-Tree Insertion / AVL Tree Insertion / Red-Black Trees / Huffman Encoding Tree** | **10** | **10** |  |
| **3 a) Dijkstra’s shortest path code + test** | **5** | **5** |  |
| **3 b) Prim’s MST code + test** | **5** | **5** |  |
| **3 c) BFT or DFT code + test** | **5** | **3** | **Normal graph için yaptık** |
| **3 d) Filling Big-O Table** | **5** | **4** | **Yanlış sonuç çıkabilir fakat kaynaklarda genelde bu şekilde gözüküyor** |
| **4 i) Graph Drawing** | **5** | **3** | **Sağ alttaki iki oku ayıramadık gerekli özelliği de verdik fakat çalıştıramadık** |
| **4 ii) Finding Shortest Paths with Dijkstra’s** | **5** | **5** |  |
| **4 iii) Node deletion and repeating i, ii.** | **5** | **5** |  |
| **5 a) Comparison (Prim’s & Kruskal’s Algorithm)** | **5** | **5** |  |
| **5 b) Explanations of 4 terms** | **20** | **20** |  |
| **Demo Video for Source Codes and Tests of Q2 Q3a, Q3b and Q3c.** | **5** | **5** |  |
| **Self-assessment Table** | **5** | **5** |  |
| **Total** | **100** | **95** |  |
|  |  |  |  |